André Luiz de Souza Antonieto

Automação industrial utilizando tecnologia IoT

Itajubá - MG

Novembro de 2016

André Luiz de Souza Antonieto

Automação industrial utilizando tecnologia IoT

Monografia referente ao Trabalho Final de Graduação do curso de Engenharia da Computação da Universidade Federal de Itajubá

Universidade Federal de Itajubá – UNIFEI

Instituto de Engenharia de Sistemas e Tecnologia da Informação

Engenharia de Computação

Orientador: Rodrigo Maximiano Antunes de Almeida

Itajubá - MG

Novembro de 2016

André Luiz de Souza Antonieto
Automação industrial utilizando tecnologia IoT. Itajubá - MG, Novembro de 2016.
55 p. ; 30 cm.

Orientador: Rodrigo Maximiano Antunes de Almeida

TFG (Graduação) – Universidade Federal de Itajubá – UNIFEI
Instituto de Engenharia de Sistemas e Tecnologia da Informação
Engenharia de Computação, Novembro de 2016.

1. Sistemas Embarcados 2. Raspberry Pi 3. Arduino 4. Internet das Coisas 5. Comunicação Serial 6. Controlador PID

Agradecimentos

Aos meus pais, por todo apoio, incentivo aos estudos e por sempre terem acreditado no meu potencial para me tornar um engenheiro.

Aos amigos e colegas de faculdade que me deram suporte, companhia e muitos momentos de alegria durante a graduação.

Ao professor Rodrigo Maximiano Antunes de Almeida pela orientação no trabalho que foi fundamental para a execução e sucesso do projeto.

Aos professores do curso de engenharia de computação que deram o seu melhor transmitindo os seus conhecimentos para minha formação.

"Há mais pessoas que desistem do que pessoas que fracassam.
(Henry Ford)

Resumo

Este presente trabalho final de graduação tem como objetivo enviar ações para um controlador através de uma comunicação serial de um sistema embarcado envolvendo tecnologia IoT (*Internet of Things*). Sistemas embarcados são sistemas com funções e recursos projetados especificamente para um determinado fim. O sensoriamento embarcado tem como objetivo medir, identificar, avaliar a sua situação ou a do meio onde se encontra. Os dois termos foram utilizados nesse projeto para que seja realizada a automação industrial. Para realizar a automação é necessária a comunicação serial, que é a transferência de dados bit a bit de um sistema para o outro. Essa transferência de dados foi realizada com a ajuda da tecnologia IoT, cujo objetivo é enviar informações via protocolo web para uma placa controladora que foi responsável em manipular um controlador digital utilizando o conceito de tempo real de execução. Ao final, obteve-se um sistema completo capaz de manipular um meio conforme for programado.

Palavras-chaves: Sistemas Embarcados, Internet das Coisas, Comunicação Serial, Automação, Arduino, Raspberry Pi, Controlador Digital, Tempo Real.

Abstract

This final project was developed to send commands to a controller through serial communication using an embedded system and IoT (Internet of Things) technology. Embedded systems are systems with functions and resources projected specifically for a particular purpose. The embedded sensing is a system property that measure, identify and evaluate itself or its environment situation. Both concepts are used in this project to make the industrial automation possible. To make the industrial automation it is necessary to use the serial communication, which is the data transfer bit by bit from one system to other. This data transfer is performed with IoT technology, that purposed was send information using web protocol for a controller board which was responsible to manager a digital controller that used real time of execution. At the end, we obtained a complete system capable of manipulate as it was programmed.

Key-words: Embedded Systems, Internet of Things, Serial Communication, Automation, Arduino, Raspberry Pi, Digital Controller, Real Time.

Lista de ilustrações

Figura 1 – Exemplo de controle em Malha Aberta. 16
Figura 2 – Exemplo de controle em Malha Fechada. 16
Figura 3 – Equação do Controlador PID. 17
Figura 4 – Diagrama do Controlador PID. 17
Figura 5 – Raspberry Pi 2. 22
Figura 6 – Arduino. 23
Figura 7 – Representação do *Duty Cicle* do PWM. 24
Figura 8 – Representação da transmissão em UART 25
Figura 9 – Dispositivos e Tecnologia da Informação convergindo para IIoT. 27
Figura 10 – Diagrama do Sistema de Automação. 28
Figura 11 – Interface Web. 34
Figura 12 – Interface Web com resposta da placa de controle. 35
Figura 13 – Mensagem Enviada em ASCII. 36
Figura 14 – Mensagem enviada em ASCII representada em código binário. 36
Figura 15 – Resposta ao servidor na tentativa de um comando inválido. 37
Figura 16 – Diagrama esquemático do sistema utilizando o circuito RC. 37
Figura 17 – Curva de descarga do capacitor sem controlador. 38
Figura 18 – Curva de descarga do capacitor com controlador. 38
Figura 19 – Curva de carga do capacitor com controlador. 39
Figura 20 – Ciclo de pulsos do Tempo Real. 40

Lista de tabelas

Tabela 1 – Possibilidade de Comandos 30
Tabela 2 – Exemplo de Comando Enviado 31

Lista de Códigos

3.1 Configuração da Comunicação Serial 30
3.2 Habilitar Porta Digital do Arduino 31
3.3 Calculo do MD5 no arduino 32
3.4 Controle Digital PID 32
3.5 Configuração do TimerOne 33

Lista de abreviaturas e siglas

IoT	Internet of Things
IIoT	Industrial Internet of Things
MD5	Message-Digest algorithm 5
CPU	Central Processing Unit
GPS	Global Positioning System
PID	Proporcional Integral Derivativo
NMEA	National Marine Electronics Association
CSS	Cascating Style Sheet
PWM	Pulse Width Modulation
PHP	Personal Home Page
UART	Universal Asynchronous Receiver/Transmitter

Sumário

1 INTRODUÇÃO 13
1.1 Objetivo 13
1.2 Organização do Trabalho 14

2 REVISÃO BIBLIOGRÁFICA 15
2.1 Automação Industrial 15
2.1.1 Controlador PID 16
2.1.2 Controlador PID Digital 18
2.2 Sistemas Embarcados 20
2.2.1 Tempo real 21
2.3 *Hardware* 21
2.3.1 Raspberry Pi 21
2.3.2 Arduino 22
2.3.3 Saída PWM 23
2.4 Comunicação Serial 24
2.4.1 MD5 25
2.5 Servidor web 25
2.6 IoT (*Internet of Things*) 26

3 DESENVOLVIMENTO 28
3.1 Servidor Web 29
3.1.1 Comunicação Serial 29
3.2 Protocolo de comunicação 30
3.3 Placa de controle 31
3.3.1 Parser do protocolo 31
3.3.2 Controle Digital 32

4 RESULTADOS 34
4.1 Interface Web 34
4.2 Comunicação Serial 35
4.3 Controle Digital 37
4.3.1 Tempo Real 39

5 CONCLUSÃO 41
5.1 Trabalhos Futuros 41

REFERÊNCIAS **43**

ANEXO A – CÓDIGO DA INTERFACE WEB **46**

ANEXO B – CÓDIGO FONTE DA PLACA CONTROLADORA . . **51**

1 Introdução

A automação de processos tem como objetivo realizar o controle de sistemas de modo automático para manipulação de diferentes processos e máquinas em uma indústria (BREI, 2013).

Com o advento da internet e das telecomunicações, cada vez mais é interessante que os dispositivos estejam conectados para trocarem informações entre si, para tornar o ambiente em que se encontram mais dinâmico, com a possibilidade de reação rápida a qualquer mudança de cenário inesperada.

Este desenvolvimento culminou no conceito de IoT, responsável por conectar dispositivos a uma rede para que eles possam enviar e receber dados entre si via web.

Nos últimos anos tem-se visto a ideia de IoT adentrar também no campo da indústria, passando a se chamar IIoT(*Industrial Internet of Things*. Com a IIoT foi possível que muitas tomadas de decisões em um chão de fábrica fossem tomadas mais rapidamente, afim de garantir melhorias contínuas de velocidade, segurança e confiabilidade dos processos (MADEIRA, 2016).

1.1 Objetivo

Neste trabalho objetiva-se desenvolver um sistema de IIoT de modo robusto. Para isso, o projeto foi dividido em duas frentes: uma placa controladora com capacidade de execução de algoritmos em tempo real e uma placa com capacidade de prover um servidor web para realizar a interface da comunicação do sistema de controle com a internet. Os objetivos do projeto especificamente são:

- Desenvolver uma página web para realizar interface com o cliente e o processo de controle de uma tarefa.
- Utilizar protocolos seguros na comunicação serial para garantir confiabilidade nos dados.
- Processar os dados recebidos pela comunicação serial.
- Executar os comandos de controle estabelecidos pelo cliente na página web.
- Garantir tempo real do controlador.

1.2 Organização do Trabalho

Este documento está organizado em cinco capítulos. O segundo capítulo apresenta uma literatura aos temas abordados durante o desenvolvimento do trabalho. No terceiro capítulo, temos o desenvolvimento do projeto e as soluções para os problemas apresentados. No quarto capítulo, são demonstrados os resultados obtidos pelos testes de comunicação e controle abordados no desenvolvimento do projeto. Ao final, no quinto capítulo, a conclusão do trabalho, tomando-se em conta os resultados obtidos durante o desenvolvimento do projeto.

2 Revisão Bibliográfica

2.1 Automação Industrial

Automação é a substituição do trabalho humano ou animal por uma máquina. Algo automático significa ter um mecanismo de atuação própria, que faça uma ação requerida em tempo determinado ou em resposta a certas condições. O conceito de automação inclui a ideia de se usar energia, seja ela mecânica ou elétrica para acionar algum tipo de máquina. A máquina precisa ter algum tipo de inteligência para que ela execute sua tarefa de modo mais eficiente, seguro e econômico (RIBEIRO, 1999).

O objetivo principal da automação industrial é criar sistemas para melhorar a produção com o menor custo para a empresa. Alguns objetivos da automação industrial são: Melhorar a produtividade, melhorar as condições de trabalho aumentando a segurança, realizar operações que seria praticamente impossíveis manualmente, simplificar a operação e manutenção de modo que não seja necessário grande conhecimento para manuseio do sistema (SILVEIRA, 2013).

Com o aumento da produção, qualidade dos produtos, exigências de segurança e necessidade de reduzir os custos de produção, os sistemas industriais estão necessitando cada vez mais de máquinas automatizadas.

Uma das linhas da automação de sistemas é o controle de grandezas como temperatura, pressão, entre outras.

Um sistema de controle é composto por um ou mais dispositivos que comandam o comportamento de outros dispositivos. Permite que uma determinada referência seja seguida pela variável obedecendo alguns critérios, seja tempo de acomodação, erro em regime permanente, consumo de energia, entre outros. Um sistema de controle pode ser classificado como:

- **Malha Aberta**: Esse sistema executa ações pré-definidas sem possibilidade de correção dessas ações caso o sistema não se comporte caso esperado. É um sistema de controle sem realimentação como demonstra a figura 1 (SILVA, 2000b).
- **Malha Fechada**: Esse sistema também executa ações pré-definidas, porém está sempre sendo comparado com o comportamento esperado, tem capacidade de fazer correções automáticas do sistema para tentar atingir o estado desejado. É um sistema de controle com realimentação como demonstra a figura 2 (SILVA, 2000c).

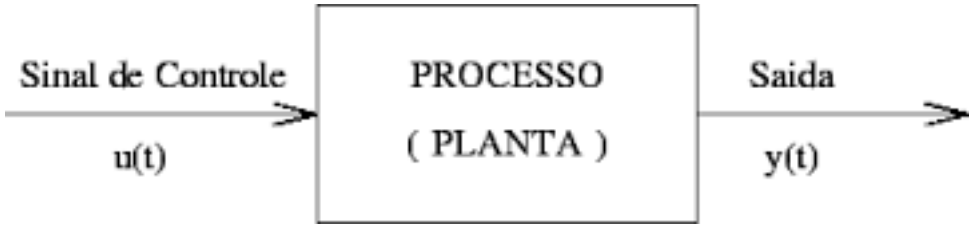

Figura 1 – Exemplo de controle em Malha Aberta.

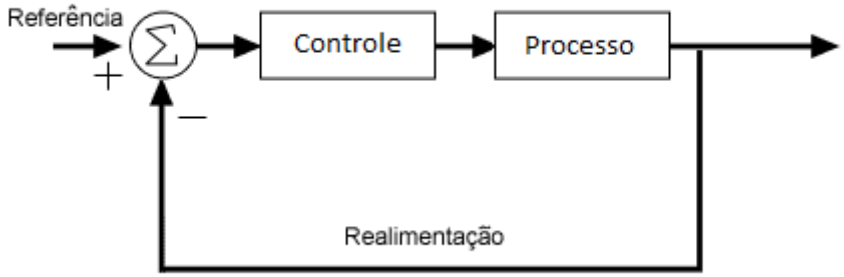

Figura 2 – Exemplo de controle em Malha Fechada.

2.1.1 Controlador PID

O Controlador proporcional integral derivativo ou PID é um controlador muito usado na indústria pois possui excelente desempenho e pode ser acessado por outros programas que precisem enviar comandos ou receber informações. A ideia básica de um controlador PID é ler um sensor e calcular a resposta de saída. No sistema, ele fará interface com o Arduino, onde receberá comandos e enviará informações se as ações requisitadas foram executadas (INSTRUMENTS, 2011).

O funcionamento desses controladores se baseia no cálculo inicial do erro e(t) entre a variável controlada e seu valor desejado. O algoritmo do PID usa o erro em três módulos distintos (proporcional, integral e derivativo, que dá o nome ao algoritmo) para produzir o sinal de saída de forma a estabilizar e manter estável o sistema da melhor forma possível (FELIPE, 2014).

O sinal de controle u(t) pelo controlador PID é dado por:

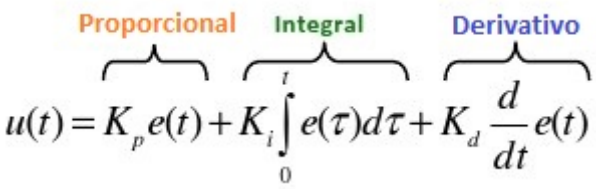

Figura 3 – Equação do Controlador PID.

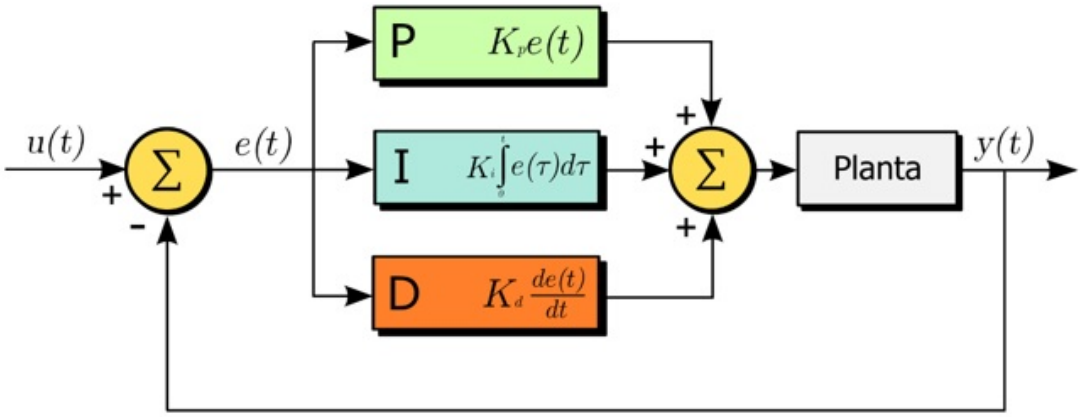

Figura 4 – Diagrama do Controlador PID.

A resposta proporcional da função do controlador PID pode ser ajustada a partir da constante de ganho Kp, ela produz um valor na saída de acordo com o erro obtido na realimentação. Quanto maior for essa constante, maior será o ganho do erro e mais instável será o sistema. Porém, se a constante Kp for pequena, o tempo de resposta do sistema será menor (OGATA, 2011).

A função integral soma todos os erros instantâneos. Essa soma é multiplicada pela constante Ki. A função integral do controlador acelera o movimento do processo até o ponto desejado e elimina o erro que ocorre na função anterior. Como a função soma dados instantâneos, o resultado do processo pode ultrapassar o ponto desejado. Essa consequência se chama *overshoot* (OGATA, 2011).

A função derivativa diminui a taxa de variação de saída do controlador. Essa função diminui o *overshoot*"da função anterior e melhora a estabilidade do controlador. Por outro lado, a função derivativa causa um atraso na resposta e é muito sensível à ruídos. Isto se deve porque essa função aumenta o ruído e caso o ruído e o ganho Kd forem muito grandes, podem causar instabilidade no controlador (OGATA, 2011).

2.1.2 Controlador PID Digital

Um controlador digital trabalha com sinais numéricos e é fisicamente implementado como uma rotina ou programa a ser executado sobre um micro controlador. São utilizados para medição e controle de temperatura, ligar/desligar motores, ler um teclado em sua entrada, etc (COSTA, 2006).

O Controle digital de um processo envolve basicamente cinco etapas: amostragem, conversão analógica/digital, cálculo do controle através de um programa, conversão digital/analógica e aplicação do sinal de controle calculado até o próximo instante da amostragem (SILVA, 2000a).

Para se utilizar o controle digital PID foi necessário utilizar o método de Tustin ou aproximação bilinear, que permite transformar uma função analógica em digital. O método da aproximação bilinear é uma técnica que mapeia polos e zeros de uma função contínua no tempo para um sistema discreto (NEVES, 2015).

A técnica descrita abaixo utiliza a aproximação bilinear, uma transformação entre as variáveis z que mapeia todo eixo-j no plano-s para uma revolução unitária no plano-z.Com Hc(s) denotando a função do sistema de tempo contínuo e H(z) a função do sistema de tempo discreto, a transformação bilinear corresponde à substituição de s por (OPPENHEIM; SCHAFER, 1989):

$$s = \frac{2}{T}.\frac{z-1}{z+1} \tag{2.1}$$

Onde T é o período de amostragem do sinal.

Mapeando s em z, podemos definir a transformação bilinear como (OPPENHEIM; SCHAFER, 1989):

$$H(z) = H_c(s)\Big|_{s=\frac{2}{T}\frac{z-1}{z+1}} = H\left(\frac{2}{T}\frac{z-1}{z+1}\right). \tag{2.2}$$

Um controlador PID possui a seguinte equação característica:

$$G_c(s) = K_p + K_d.s + \frac{K_i}{s} \tag{2.3}$$

Sendo Kp o coeficiente da ação proporcional, Ki o coeficiente da ação integral e Kd o coeficiente da ação derivativa.

A partir desta, obtém-se a sua forma digital através do uso da transformada-Z e algumas manipulações algébricas mostradas a seguir.

Primeiro, aplica-se a transformação bilinear, simplesmente fazendo a substituição:

$$s = \frac{2}{T}.\frac{z-1}{z+1} \tag{2.4}$$

Após a substituição, a equação de controle em Z é mostrada abaixo:

$$\frac{U(s)}{E(s)} = k_p + \frac{2}{T}\left(\frac{z-1}{z+1}\right)k_d + \frac{T}{2}\left(\frac{z+1}{z-1}\right)k_i \tag{2.5}$$

$$\frac{U(s)}{E(s)} = \frac{k_p.2T(z+1)(z-1) + (2(z-1))^2.K_d + (T(z+1))^2.K_i}{T(z+1).2(z-1)} \tag{2.6}$$

Sendo U(s) o sinal de saída do sistema e E(s) o sinal de erro na entrada do controlador.

Multiplicando cruzado os termos dos dois lados:

$$\begin{aligned} U(z)z^2 - U(z) = (e(z).k_p.z^2 - e(z).k_p) + \\ \tfrac{2}{T}(e(z).z^2.k_d - 2.e(z)z.k_d + e(z).k_d) + \\ \tfrac{T}{2}(e(z).z^2.k_i + 2.e(z).z.k_i + e(z).k_i) \end{aligned} \tag{2.7}$$

Multiplicando todos termos por z^{-2}, fica:

$$\begin{aligned} U(z) = U(z).z^{-2} + (e(z).k_p - e(z).k_p.z^{-2}) + \\ \tfrac{2}{T}(e(z).k_d - 2.e(z)z^{-1}.k_d + e(z).z^{-2}.k_d) + \\ \tfrac{T}{2}(e(z).k_i + 2.e(z).z^{-1}.k_i + e(z).z^{-2}.k_i) \end{aligned} \tag{2.8}$$

Depois de obter a equação discreta, é preciso aplicar a transformada inversa de z, que transforma a equação no domínio discreto para uma equação diferencial linear, pois equações algébricas possuem soluções mais fáceis. A transformada inversa de z em uma forma complexa é dada por (JUNIOR; CRUZ, 2010)(OPPENHEIM; SCHAFER, 1989):

$$x[n] = \mathcal{Z}^{-1}\{X(z)\} = \frac{1}{2\pi j}\oint_C X(z)z^{n-1}dz \tag{2.9}$$

Onde C representa um contorno fechado dentro da região de convergência da transformada z.

Para retornar do domínio da frequência discreta Z, para o domínio do tempo discreto N podemos utilizar a seguinte correlação (OPPENHEIM; SCHAFER, 1989):

$$z^{-k}X(z) = X(n-k) \tag{2.10}$$

Onde n é o n-ésimo instante de amostragem e k a variação que possibilita comparar com um amostragem diferente do instante n.

Aplicando a transformada Z inversa e usando a correlação acima, temos como resultado:

$$\begin{aligned} U(n) = U(n-2) + e(n).k_p - e(n-2).k_p + \\ \frac{2}{T}(e(n).k_d - 2.e(n-1).k_d + e(n-2).k_d) + \\ \frac{T}{2}(e(n).k_i + 2.e(n-1).k_i + e(n-2).k_i) \end{aligned} \tag{2.11}$$

Onde n é o n-ésimo instante de amostragem.

No código abaixo temos a transforma inversa de z para o controlador PID, as variáveis e0, e1 e e2 são responsáveis pelo erro do controlador. As variáveis y0, y1 e y2 são as saídas do controlador. Como é necessário ter o histórico das entradas e saídas, utilizamos essas seis variáveis para manipular esse histórico de dados.

Observa-se ao final que a saída do sinal do controlador é dependente da entrada atual y2, das 2 últimas entradas y1 e y0 . Assim, é necessário guardar valores de entrada e saída durante a execução do programa.

```
//Atualização das variáveis do estado anterior
e2 = e1;
e1 = e0;
e0 = referencia - ValorAD();

y0 = y1;
y1 = y2;

//equação do PID digital com aprox. bilinear
y2 = y0 + (kp * (e0          - e2)        ) +
          (ki * (e0 + (2*e1) + e2) * T / 2) +
          (kd * (e0 - (2*e1) + e2) * 2 / T);
```

2.2 Sistemas Embarcados

Um sistema embarcado trata-se de um sistema com capacidade computacional, de modo que este seja mais do que um simples computador, sendo um sistema completamente independente e capaz de realizar uma tarefa determinada. Um sistema caracterizado como embarcado pode ser um sistema desenvolvido com o objetivo de realizar uma tarefa específica e direcionada, onde os componentes que fazem parte do sistema variam, apesar de ser composto basicamente pelos mesmos componentes presentes em um computador pessoal (FELIPE; ALEXSANDER; LEONARDO, 2014).

Sistemas embarcados estão relacionados com *hardware* e *software* em um único dispositivo. A diferença de um sistema embarcado para um computador pessoal está na diferença de objetivos. Um computador é capaz de realizar muitas tarefas. Um sistema embarcado consegue realizar poucas tarefas ou até mesmo somente uma tarefa (DELAI, 2013).

2.2.1 Tempo real

Comunicação em tempo real é uma comunicação em que os dispositivos que a utilizam pedem requisitos e, a rede em que esses dispositivos estão conectados garante que esses requisitos foram atendidos. O tempo em que essa tarefa será executada não importa, o que importa é se os requisitos foram perfeitamente atendidos. Por exemplo, em uma fábrica de componentes químicos alguns sensores indicam os teores de uma mistura a um sistema de controle. Esse sistema de controle envia ordens para que a mistura seja sempre corrigida, adicionando material, corrigindo temperatura ou pressão. É necessário que esse sistema não sofra atrasos ou perdas no envio de informações, pois isso pode gerar sérios danos à produção (FELIPE, 2014).

Comunicação em tempo real significa muito mais do que fazer acontecer algo imediatamente após a requisição de um comando. Tempo real em computação envolve o conceito de máquinas executarem tarefas num tempo pré-definido. O tempo real é necessário para garantir que o sistema executará uma determinada tarefa no mesmo intervalo de tempo (WOOD, 2015).

2.3 *Hardware*

2.3.1 Raspberry Pi

Raspberry Pi vide figura 5, é um computador de pequeno porte que possui um processador ARMv7 e 1GB RAM com intenção de reduzir o custo dos computadores e disponibilizar *softwares* gratuitamente para estudantes. Ela é uma placa um pouco mais lenta que um computador pessoal, mas é capaz de rodar um sistema operacional Linux usando uma baixa quantidade de energia (VERRY, 2012)(FOUNDATION, 2016).

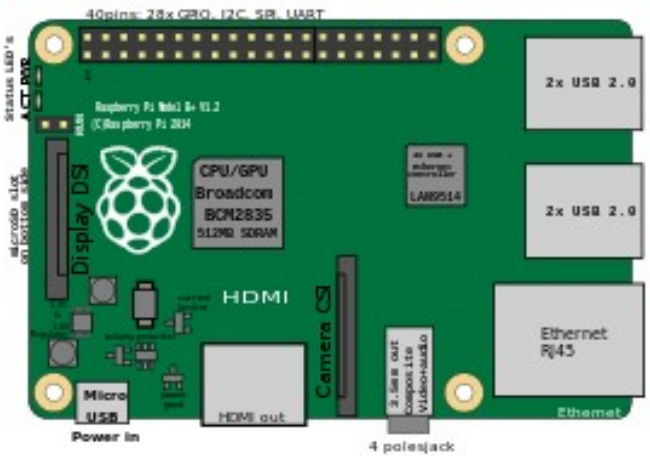

Figura 5 – Raspberry Pi 2.

2.3.2 Arduino

Segundo o fabricante, o Arduino UNO é uma placa micro controladora baseado no ATmega328. Possui 14 pinos *input/output* digitais, 6 *inputs* analógicos, um ressoador de cerâmica de 16MHz, uma conexão USB, uma tomada de energia e um botão *reset*. Contém tudo que é necessário para estabelecer o suporte necessário ao micro controlador. Para utiliza-lo basta conecta-lo a um computador por um cabo USB ou energiza-lo de um adaptador AC-DC ou bateria. A figura 6 é uma referência da placa Arduino UNO.

O Arduino pode ser programado utilizando a IDE "Arduino *Software*", o Arduino vem programado com um *bootloader* que permite fazer *upload* de códigos sem usar algum dispositivo extra de *hardware*. O Arduino possui uma grande facilidade para se comunicar com um computador, outro Arduino e outro micro controlador. Ele tem uma comunicação serial UART TTL disponível nos pinos digitais 0 (RX) e 1 (TX). O *firmware* utiliza os *drivers* padrões do USB COM. O *software* do Arduino possui um monitor serial que possibilita a transferência de dados de placa para placa (LUIZ; VINICIUS; VINICIOS, 2015).

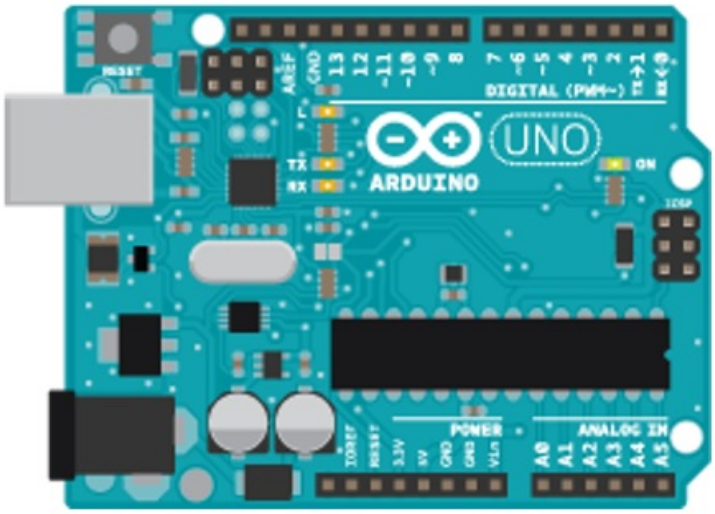

Figura 6 – Arduino.

2.3.3 Saída PWM

O PWM é uma técnica utilizada para variação do valor médio de uma forma de onda periódica. A técnica consiste em manter a frequência de uma onda quadrada e variar o tempo que o sinal fica em nível lógico alto. Esse tempo é chamado de *duty cycle*. Quando o *duty cicle* está em 0%, o valor médio da saída encontra-se em 0 V e, para um *duty cycle* de 100%, a saída tem valor máximo de 5V (SOUZA, 2014).

O Arduino Uno possui alguns pinos para saída PWM (3,5,6,9,10,11). Para auxiliar na manipulação desses pinos, a plataforma possui uma função que auxilia na escrita de valores de *duty cycle* para esses pinos. A função que manipula esses pinos se chama analogWrite().

A Figura 7 ilustra o comportamento da técnica PWM:

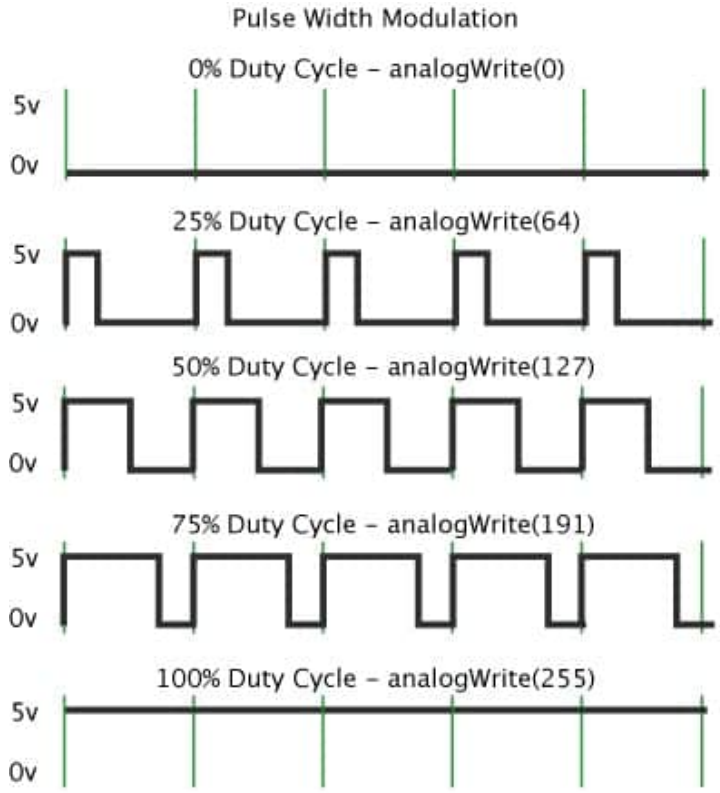

Figura 7 – Representação do *Duty Cicle* do PWM.

2.4 Comunicação Serial

A comunicação serial é a transferência de dados bit a bit, podendo ser dividida em dois grupos:

- **Síncrona**: O emissor e o receptor devem estar sincronizados, e permanecer em sincronia durante a troca de informações. A informação é transmitida em instantes de tempo bem definidos. O emissor e o receptor nunca enviam mensagens no mesmo momento, um sempre espera o outro enviar o bloco de informações por completo (FREITAS, 2016).
- **Assíncrona**: O emissor e o receptor não estão sincronizados, existe apenas bits para indicar que o início e fim do caractere transmitido. Esse tipo de transmissão é caracterizado por poder começar a qualquer momento (FREITAS, 2016).

A UART (*universal asynchronous receiver/transmitter*) é um recurso responsável por realizar a comunicação serial assíncrona. Possui duas portas: RX e TX, sendo um responsável por receber a informação e outro para transmitir a informação. A UART transmite os bits de dados individualmente em série, o receptor UART recebe todos os bits da transmissão e remonta a mensagem (JIMBO, 2012). A figura 8 demonstra a transmissão em UART.

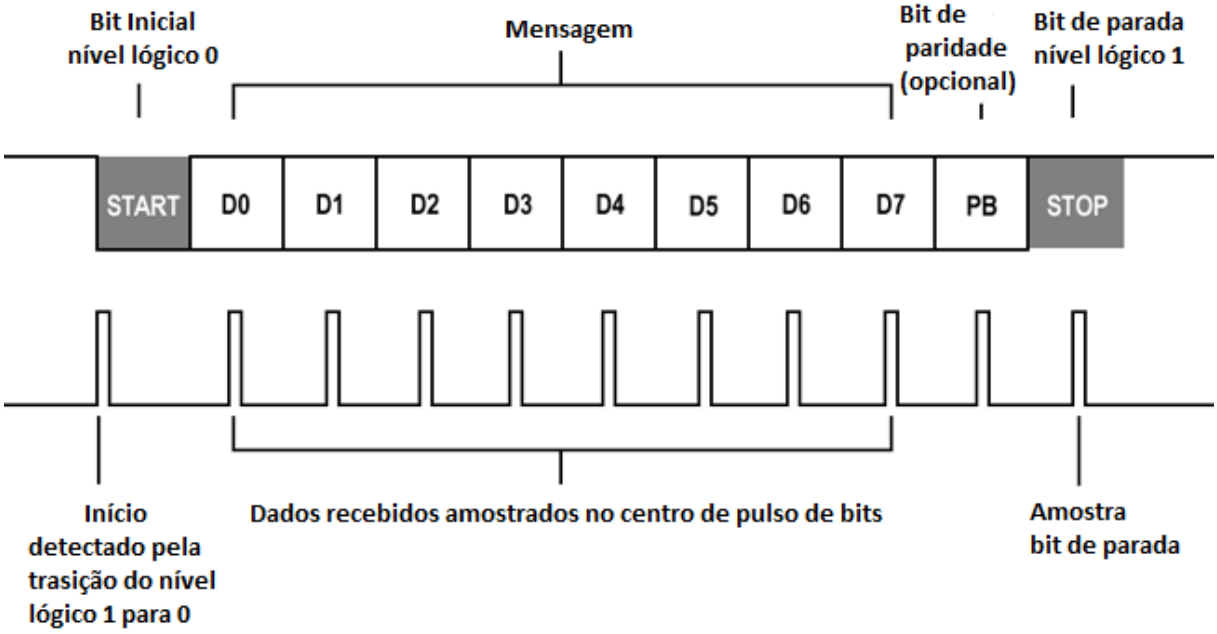

Figura 8 – Representação da transmissão em UART

Como todo canal de comunicação, durante a transferência de bits é possível ocorrer falhas, interferências do sinal. Existem alguns algoritmos que podem ajudar a evitar que sinais com falhas sejam executados. Um desses algoritmos é o MD5.

2.4.1 MD5

O MD5 é um algoritmo de *hash* de 128 bits unidirecional para verificar integridade das informações. É de domínio público para uso em geral. *Message Digests* são funções *hash* que geram códigos de uma mensagem qualquer. Nesse projeto, o MD5 será usado para gerar o *hash* da mensagem a ser enviada na comunicação serial. Esses códigos *hash* são extremamente úteis para segurança de senhas ou pacotes de mensagens. Como ele não pode ser descriptografado, o código *hash* precisa ser novamente gerado e comparado com a sequência disponível anteriormente. Se ambos se igualarem, o controlador pode executar a ação (RODARTE, 2007).

2.5 Servidor web

Um servidor web é responsável por armazenar e trocar informações com outras máquinas. Pelo menos duas máquinas estão envolvidas nessa comunicação, uma sendo o cliente, que solicita as informações, e o servidor, que atende os pedidos do cliente. Para cada máquina existe um *software* por trás para tratar essa troca de informações. O

cliente utiliza o *browser* e o servidor utiliza um *software* que trata essas informações via http (Protocolo de Transferência de Hipertexto), nesse sistema iremos utilizar o *software* chamado Lighttpd para tratar essas informações.

Uma comunicação simples entre um cliente e um servidor acontece da seguinte forma: O *browser* entra com o endereço URL onde contém o nome do domínio, nome da página e o protocolo. Por exemplo, http://192.168.1.10/teste.php. O protocolo nesse exemplo é o http, o domínio 192.168.1.10 e o nome da página teste.php. O servidor interpreta esse domínio, verifica se a página existe para esse endereço de IP e retorna a página encontrada (SILVA, 2012).

Lighttpd : O Lighttpd é um servidor web *open source* de alta performance. Possui grande velocidade, segurança e utiliza pouca memória comparado com os outros servidores web. Como o nosso sistema é embarcado, não muito rico em memória e também precisamos de uma gestão eficaz da CPU, o Lighttpd se torna uma ótima opção (AVOYAN, 2012).

2.6 IoT (*Internet of Things*)

IoT ou no português "Internet das Coisas" é um sistema que inter-relaciona equipamentos, dispositivos, objetos e pessoas. Esse sistema possibilita a transferência de dados e tomada de decisões de uma máquina ou pessoa através da internet (ROUSE, 2016).

A ideia de IoT surgiu por volta de 1991 junto com os protocolos de conexão TCP/IP. Quem usou esse termo pela primeira vez foi o empreendedor e um dos fundadores da Auto-ID Center do MIT, Kevin Ashton. Ele fazia parte de um time que descobriu como fazer um *link* de objetos com a internet usando RFID (SAS, 2016).

A Internet das Coisas possui características que a torna muito interessante para o mundo industrial como por exemplo, realizar ações de um processo industrial via internet e monitorar o mesmo a partir dos resultados enviados de volta para web. Com ela é possível ter soluções inteligentes e ganho de velocidade na produção, reduzindo o consumo de combustível. Além de tudo, todas essas informações podem ficar salvas na nuvem e podem ser acessadas de qualquer lugar do planeta(ZAMBADA, 2014).

Nos últimos anos tem sido cunhado o termo IIoT (*Industrial Internet of Things*), que visa justamente utilizar o conceito de IoT nas indústrias, inclusive culminando num consórcio de empresas com o objetivo de padronizar e agilizar a adoção destas tecnologias na indústria. As empresas podem se beneficiar com IIoT visando um crescimento futuro, pois ela poderá proporcionar aumento nas receitas, produtividade e estimulo de tecnologias mais inteligentes e inovadoras (DAUGHERTY et al., 2016).

Podemos também definir IIoT como a fusão da tecnologia da informação com tecnologia operacional (figura 9), apesar de elas terem objetivos diferentes olhadas separadamente, unidas são capazes de otimizar vário processos, melhorando também a análise de dados dos mesmos (DESAI, 2016).

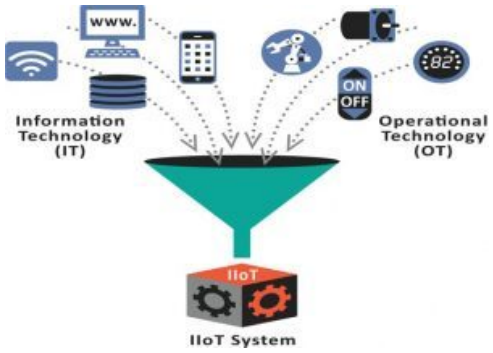

Figura 9 – Dispositivos e Tecnologia da Informação convergindo para IIoT.

3 Desenvolvimento

O Sistema foi dividido em três etapas: servidor web, protocolo de comunicação e placa de controle. Cada etapa possui funções específicas que juntas são capazes de automatizar um processo. A primeira etapa é responsável por fornecer um servidor web onde o cliente entrará com dados para realização do controle digital. O protocolo de comunicação é responsável em realizar toda comunicação serial que transfere dados de uma placa para outra utilizando alguns recursos de segurança de comunicação. A última etapa é onde o Arduino realiza o controle digital com os dados fornecidos do servidor

O Arduino e a Raspberry Pi foram os sistemas embarcados escolhidos para realizar as funções do sistema de automação industrial. A Raspberry Pi foi responsável pelo servidor web que forneceu uma página de internet onde o usuário pode escolher ações a serem enviadas para a placa de controle. A Raspberry Pi interpreta as ações escolhidas e envia via comunicação serial para o Arduino. O Arduino foi a placa de controle responsável por executar a ação recebida da Raspberry Pi e enviar uma resposta se a ação foi executada com sucesso ou não. A figura 10 é um esquemático para representar a divisão do sistema.

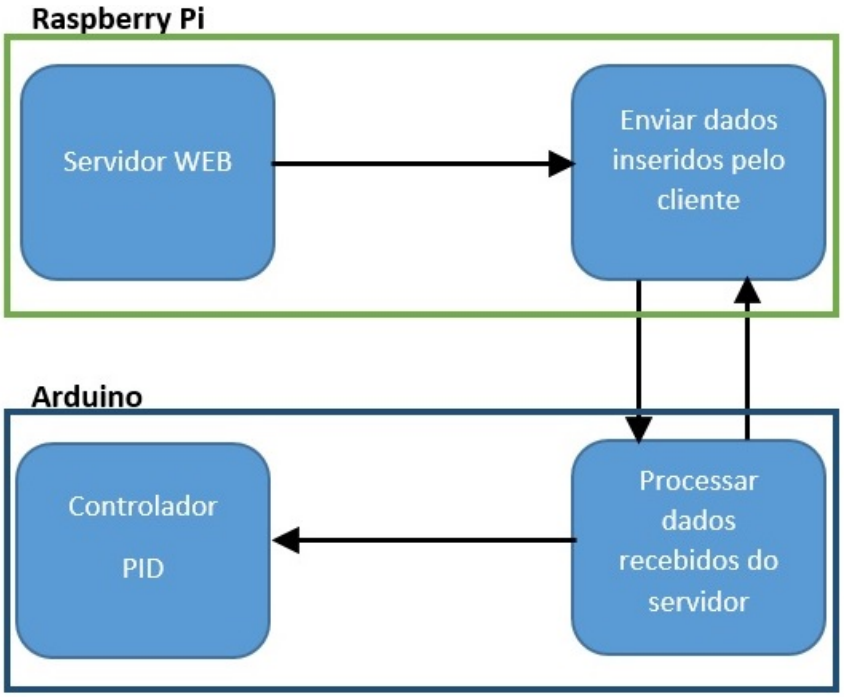

Figura 10 – Diagrama do Sistema de Automação.

3.1 Servidor Web

O primeiro passo no desenvolvimento do projeto, foi configurar o servidor na Raspberry Pi. Foi escolhido como servidor o Lighttp, pois ele é um *software* que consegue gerenciar melhor a CPU. Como foi utilizado sistemas embarcados, e estes possuem carência de memória e processador, o Lighttpd se torna um *software* interessante para essa aplicação. Para instalação destes serviços no Linux foi utilizado o gerenciador de pacotes apt-get.

Para realizar o *back-end* da interface web, foi utilizada a linguagem de programação PHP (*Personal Home Page*) pois, a biblioteca que envia os dados da web para o Arduino via comunicação serial, foi implementada em PHP.

Os arquivos que realizam a interface do servidor web ficam localizados no diretório **/www/var/html/** da Raspberry Pi. Nela contém, o arquivo principal da web e dois arquivos apoio, sendo um da biblioteca PHP-Serial e o outro arquivo de CSS que é responsável por modificar a interface do site proporcionando por exemplo: cores ao *site*, fontes diferenciadas e sombra aos botões.

É importante ressaltar que apesar de o sistema envolver conceitos de internet, ele pode ser utilizado sem conexão a mesma, utilizando-se apenas a rede local ou via *localhost* na propria Raspberry Pi.

3.1.1 Comunicação Serial

A comunicação entre o Arduino e a Raspberry Pi será feita via USB utilizando a linguagem PHP na Raspberry Pi e a linguagem de programação do Arduino. Ao conectar o USB do Arduino na Raspberry Pi, será criado um canal de comunicação normalmente nomeado como ttyACM0 (CURVELLO, 2015).

Para tratar a comunicação entre ambas as placas iremos fazer um protocolo. Um protocolo é uma regra que estabelece como a comunicação acontecerá, como ambas as partes vão entender a informação. Nesse sistema iremos utilizar o protocolo NMEA (*National Marine Electronics Association*. Ele é utilizado no GPS (*Global Positioning System* e consiste em sentenças ASCII cuja as informações são separadas por símbolos. Isso facilita o processo pois quem recebe as informações sabe precisamente a ordem em que as informações foram enviadas.

A biblioteca PHP-Serial é um código aberto em linguagem PHP desenvolvida por (SANCHEZ, 2014) que consegue acessar as portas seriais de um computador que tem como sistema operacional Linux, OS ou Windows. Para se utilizar essa biblioteca, é necessário fazer algumas configurações dos padrões de comunicação no código da interface web que podem ser vistos no código 3.1 :

```
include "PhpSerial.php"; //import da biblioteca de serial com php

$serial = new phpSerial(); //Cria novo objeto para comunicacao
$serial->deviceSet("/dev/ttyACM0");
$serial->confBaudRate(9600);
$serial->confParity("none"); //sem paridade
$serial->confCharacterLength(8); //8 bits de mensagem
$serial->confStopBits(1); //1 bit de parada
$serial->confFlowControl("none"); //sem controle de fluxo
$serial->deviceOpen();//abre o dispositivo para comunicacao
```

Código 3.1 – Configuração da Comunicação Serial

Após as configurações, existem dois comandos disponíveis para enviar e receber informações vide código abaixo;

```
$serial->sendMessage($variavelString);
$serial->readPort();
```

E depois da comunicação feita, é necessário fechar o canal de comunicação. Basta utilizar o comando:

```
$serial->deviceClose();
```

3.2 Protocolo de comunicação

Como dito anteriormente, nesse projeto utilizamos os protocolos NMEA para organização da mensagem. Nesse projeto existe a possibilidade de se enviar cinco comandos para a placa de controle:

Comandos	**Função**
Digital ON	Habilita uma porta digital da placa de controle
Digital OFF	Desabilita uma porta digital da placa de controle
MD5 False	Envia um comando de teste com MD5 errado para exemplificar que se o der falha na comunicação, o controlador não executa a ação
GET	Retorna os valores de Kp, Ki e Kd para o usuário na tela
Submit	Enviar os valores de Kp, Ki, Kd e a referência para a placa de controle

Tabela 1 – Possibilidade de Comandos

Utilizando o protocolo NMEA, a mensagem enviada é dividida em partes. A primeira parte é para o reconhecimento da placa de controle que alguma requisição de comando foi enviada. A segunda parte é o comando. A última parte é o MD5 da mensagem enviada. Nessa última parte o MD5 pega toda a mensagem e não só o comando, e além disso tem uma senha adicional que é uma variável definida no código PHP e também no Arduino que tem que ser adicionada durante o cálculo do MD5. Essa chave é uma forma de assegurar que caso a mensagem do canal fosse interceptada, quem interceptou não conseguiria gerar o MD5 dessa mensagem pois não teria essa senha extra, garantindo um pouco mais de segurança no sistema. Abaixo segue o exemplo da mensagem que habilita uma porta digital do Arduino e como foi implementado utilizando o código PHP:

$TFG	,	digitalON	#	*MD5* ($TFG,digitalON# + tfgECO2016)	\n

Tabela 2 – Exemplo de Comando Enviado

```
1 $salt = "tfgECO2016";
2 $cmd = "digitalON";
3 $str = "$" . "TFG" . "," . $cmd . "#";
4 $strmd5 = $str . md5($str . $salt) . "\n";
5 $serial->sendMessage($strmd5);
```

Código 3.2 – Habilitar Porta Digital do Arduino

3.3 Placa de controle

O Arduino como placa de controle, tem algumas responsabilidades como entender a mensagem recebida, já que ela chega com um protocolo, deve verificar se a mensagem recebida é válida e realizar o controle digital utilizando tempo real.

3.3.1 Parser do protocolo

O Arduino recebe caractere por caractere da Raspberry Pi e analisa cada um deles. Primeiramente verifica se o primeiro caractere é $, já que é o primeiro caractere enviado da mensagem. Caso seja ele continua interpretando o resto da mensagem, se não ignora a mesma e espera receber outra mensagem. Segundo, após receber o caractere ","ele se prepara para concatenar o comando que deve ser realizado. E por fim ao receber o caractere "#", ele concatena o código MD5 para fazer a análise futura da veracidade do comando.

Para realizar essa interpretação de cada caractere recebido, foi utilizado uma lógica de manipulação de uma cadeia de caracteres que pode ser visualizado no código do controle no Arduino pela função *StringSerial()* disponível no anexo B.

Após dividir o comando e o MD5 da mensagem, é preciso verificar se o comando recebido está correto, para isso o Arduino calcula o MD5 desse comando utilizando a mesma cadeia de caracteres enviada do PHP incluindo a chave adicional de segurança. Para se calcular o MD5 foi utilizada um biblioteca *open source* desenvolvida por (GEORGITZIKIS, 2015). O código implementado para cálculo do MD5 e verificar se a mensagem está correta foram:

```
1 unsigned char *hash=MD5::make_hash(pkgmsg);
2 char *md5 = MD5::make_digest(hash,16);
3
4 flag = true;
5 for(int i=0; i<32; i++){
6    if(pkgmd5[i] != md5[i]){
7        flag = false;
8    }
9 }
```

Código 3.3 – Calculo do MD5 no arduino

3.3.2 Controle Digital

O controle digital é realizado pela seguinte código no Arduino:

```
1  void PID(void){
2    int an = analogRead(0);
3    digitalWrite(8,HIGH);
4    e2=e1;
5    e1=e0;
6    e0=ref-(an/4);
7    y0=y1;
8    y1=y2;
9    y2 = y0 + (kp * (e0 -e2)) + (ki * (e0 + (2*e1) + e2) * T/2) + (
       kd * (e0 - (2*e1) + e2) * 2/T);
10
11   if(y0 > 255){
12     analogWrite(ledPin,255);
13   }else if(y0 < 0){
14     analogWrite(ledPin,0);
15   }else{
16     analogWrite(ledPin,y0);
17   }
18   digitalWrite(8,LOW);
```

```
19 }
```

Código 3.4 – Controle Digital PID

A função apresentada no código 3.4, tem como saída a variável y0 que pode jogar um valor para a porta de saída do Arduino PWM com um valor de 0 a 255, que é convertida no valor em volts de zero a cinco. Essa função é chamada a cada um milissegundo, isso foi possível com o uso de uma biblioteca do Arduino chamada *timerone*.

A biblioteca *timerone* foi responsável por garantir o tempo real de execução do controle digital. Ela gera uma interrupção na execução do Arduino para sempre executar uma função pré-definida, que no projeto era a função PID. A configuração da biblioteca pode ser feita seguindo o código 3.5 abaixo:

```
1  Timer1.initialize(1000); //Define o periodo em microsegundos
2  Timer1.attachInterrupt(PID); //Define a funcao a ser chamada em
       cada interrupcao da biblioteca
```

Código 3.5 – Configuração do TimerOne

A biblioteca *timerone* utiliza duas saídas (9 e 10) do PWM para gerar a interrupção, então é importante que essas duas saídas não sejam utilizadas pelo controlador digital para gerar a saída y0.

4 Resultados

Esse capítulo tem por objetivo demonstrar em forma de discussões, imagens e gráficos os resultados obtidos por meio do sistema de automação desenvolvido e já apresentado.

4.1 Interface Web

A figura 11 demonstra a interface web em que o usuário pode selecionar comandos a serem enviados para o sistema de controle:

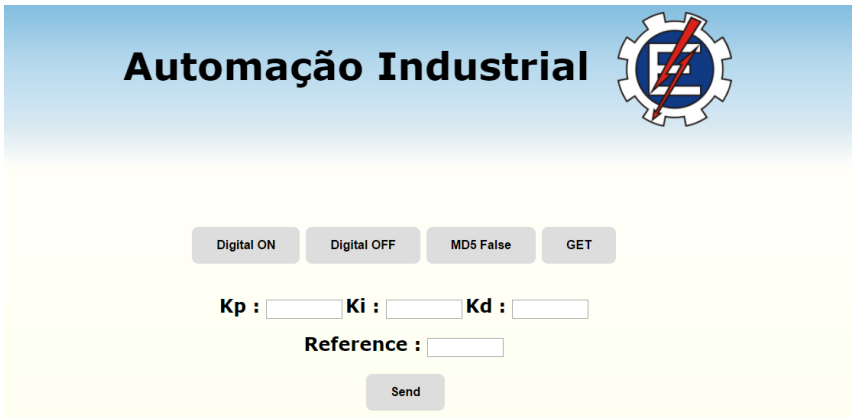

Figura 11 – Interface Web.

As funções comandos disponíveis na interface podem ser verificados na tabela 1.

Na interface web também foi implementada a lógica para demonstrar dados enviados da placa de controle via comunicação serial para o servidor. A figura 12 é um exemplo de como esses dados aparecem na página web. A mensagem em vermelho na página web, é a mensagem completa que o sistema de controle processou e enviou de volta para o servidor web.

Figura 12 – Interface Web com resposta da placa de controle.

4.2 Comunicação Serial

Ao enviar um comando da web para o Arduino, com o auxílio de um osciloscópio (Tektronix MDO4054-3 Mixed Domain Oscilloscope) foi possível interceptar a comunicação serial e enxergar como os dados chegam ao Arduino. Foi obtido exatamente toda a mensagem que pode ser vista em ASCII e código binário vide figuras 13 e 14. É interessante ressaltar que o MD5 envia a mensagem em código ASCII para não ter a possibilidade de conflito quando um caractere for um número.

Para conferir a confiabilidade da transmissão de dados, o botão MD5 False é a tentativa de se realizar o comando de habilitar uma porta digital do Arduino, porém ao se realizar o cálculo do MD5, foi alterado um bit. Assim quando o Arduino fez o cálculo do MD5 da mensagem de habilitar a porta digital, o comando foi inválido pois, os valores do MD5 estavam diferentes. A figura 15 demonstra a resposta que o Arduino envia para o servidor caso exista problema na comunicação serial envolvendo o MD5.

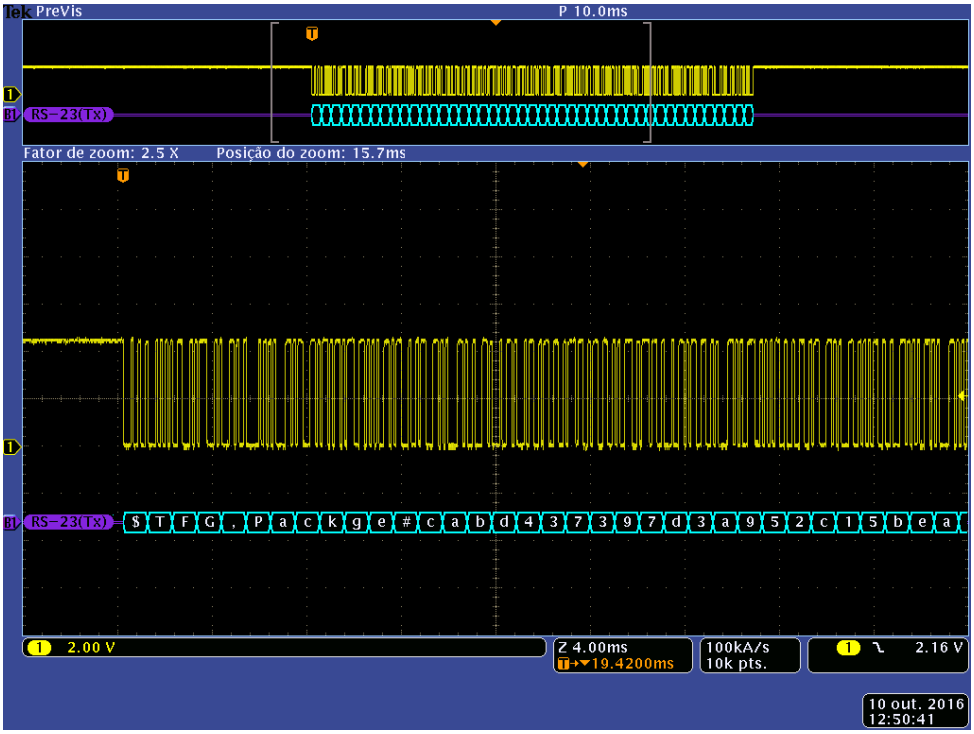

Figura 13 – Mensagem Enviada em ASCII.

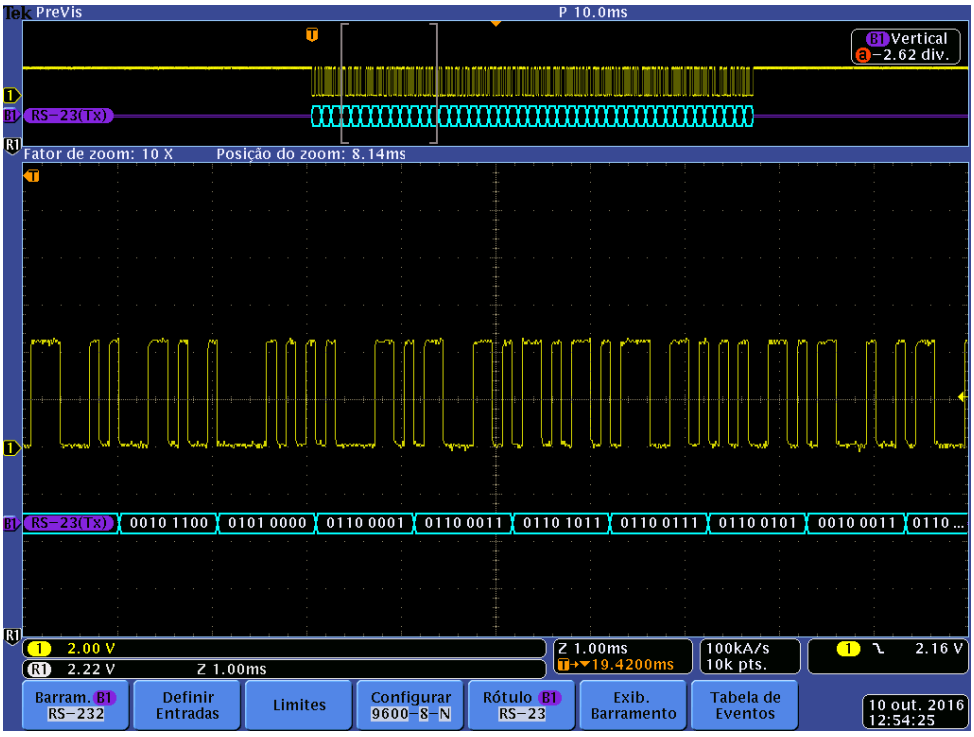

Figura 14 – Mensagem enviada em ASCII representada em código binário.

Figura 15 – Resposta ao servidor na tentativa de um comando inválido.

4.3 Controle Digital

Para realizar o controle digital, o cliente tem a possibilidade de entrar com os valores de Ki, Kp, Kd e a referência na interface web. Para testar o controle digital, foi utilizado um circuito RC representado pela figura 16, possuindo um resistor de 10 quiloohm e um capacitor de 100 microfarad . A figura 17, representa a curva de descarga do capacitor sem interferência do controlador PID digital. A figura 18, também representa a curva de descarga do capacitor. Porém podemos observador a atuação do controlador digital fazendo com que a descarga aconteça de forma mais rápida.

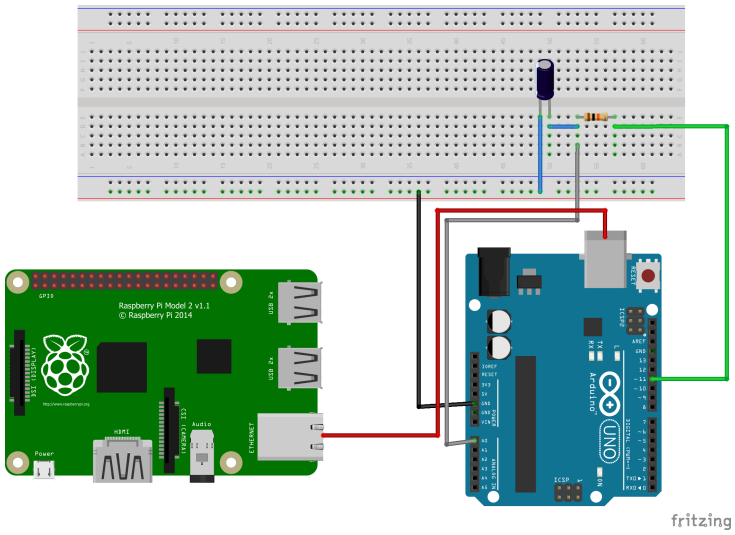

Figura 16 – Diagrama esquemático do sistema utilizando o circuito RC.

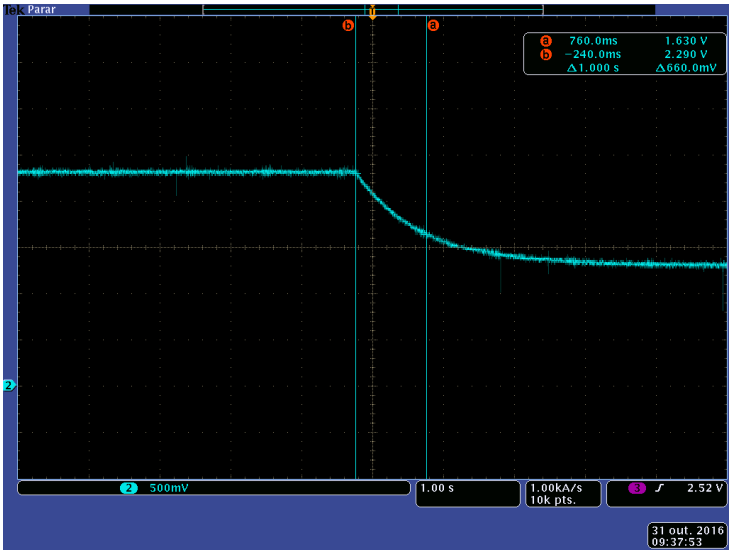

Figura 17 – Curva de descarga do capacitor sem controlador.

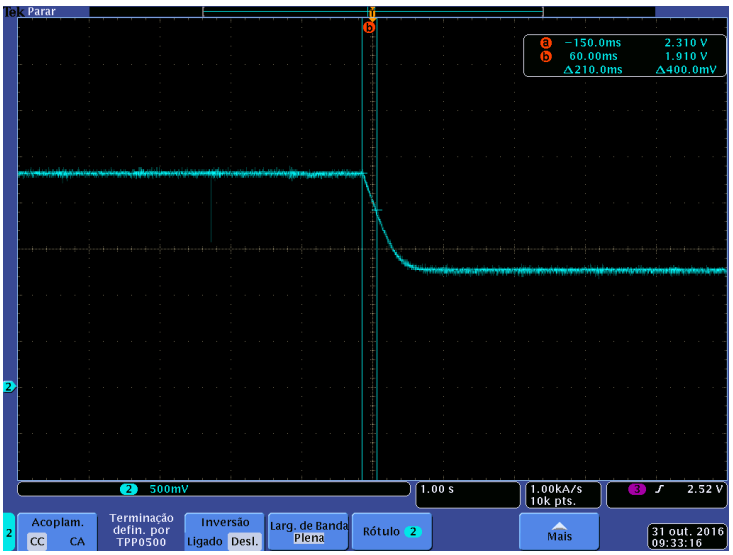

Figura 18 – Curva de descarga do capacitor com controlador.

Para o uso do controlador foram utilizados kp = 5, ki = 1, kd = 0 e referência = 80. Analisando as duas imagens observa-se a diferença nos tempos de descarga do capacitor. Na figura 17 o capacitor demora quatro segundos para descarregar completamente e na figura 18 o tempo de descarrega é de um segundo.

Também foi realizado testes para carga do capacitor, utilizando valores kp = 1, ki = 10, kd = 0 e referência = 150. Foi possível perceber um *overshoot* na carga do capacitor representado pela figura 19 demonstrando que o controlador PID está atuando no sistema.

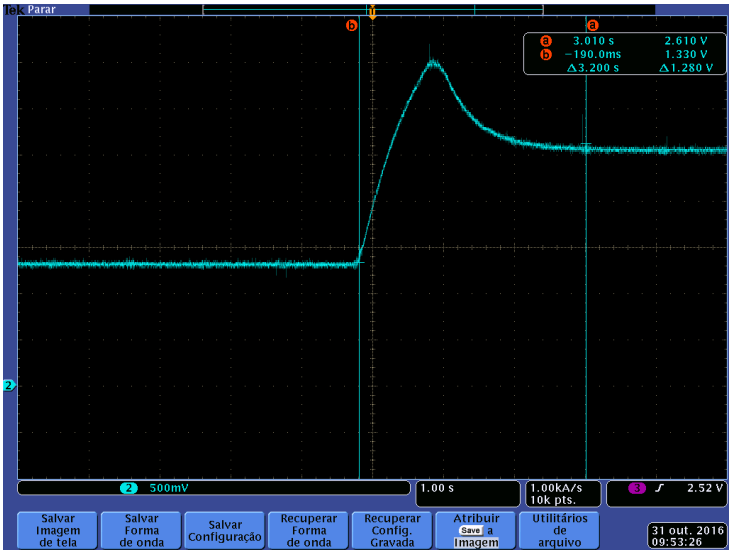

Figura 19 – Curva de carga do capacitor com controlador.

4.3.1 Tempo Real

Afim de testar o tempo real, foi utilizado uma medição no tempo de execução da função PID utilizado na placa de controle pela porta de saída 8 do Arduino. Durante a execução da função foi utilizado os comandos:

```
digitalWrite(8,HIGH);
digitalWrite(8,LOW);
```

De acordo com a figura 20, tem-se que o Arduino gera a interrupção a cada um milissegundo com um pequeno erro de no máximo 16 microssegundos, ou seja, o tempo real é garantido tendo-se um erro muito pequeno. Também é possível analisar que o Arduino leva em média 127 microssegundos ou 12,7% do tempo total de interrupção para executar a função de controle do PID. Os outros 77% são suficientes para o Arduino executar outros processos como a tarefa de processar os dados da comunicação serial.

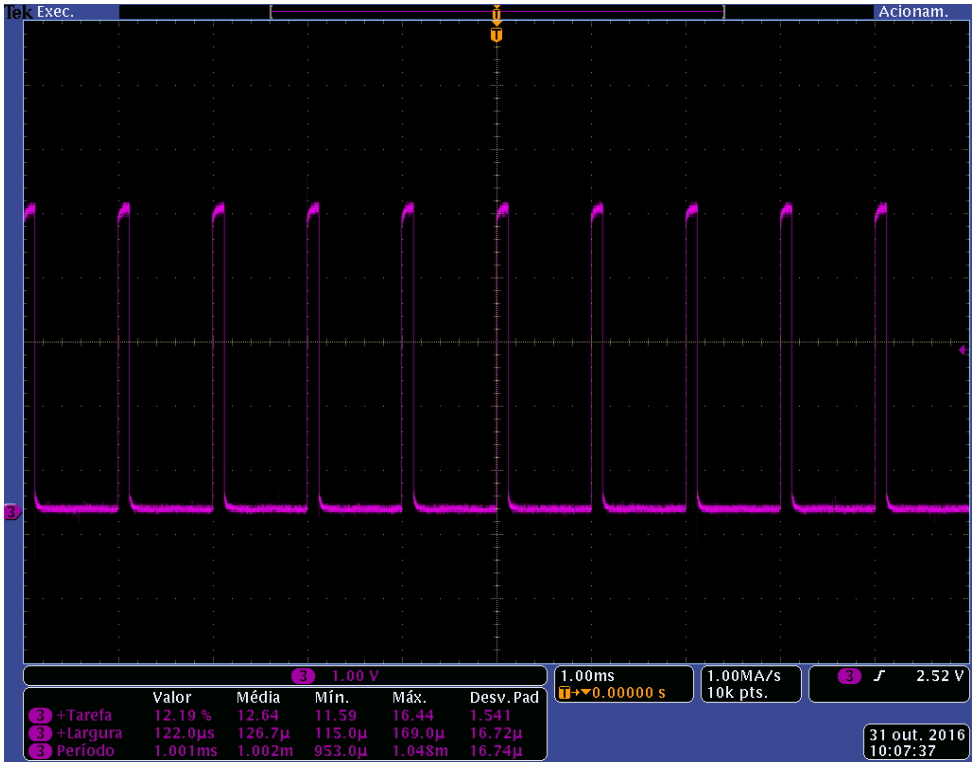

Figura 20 – Ciclo de pulsos do Tempo Real.

Caso ocorra algum erro no sistema web ou a Raspberry Pi pare de funcionar, o sistema de controle continua funcionando sem problemas. Os sistemas não são dependentes para se manterem ligados e em operação, o que ficará restrito será apenas envio de novos comandos para o sistema de controle até que o servidor web seja ligado novamente. Se houver perca de conexão com a internet o sistema ainda pode ser controlado e comandos enviados via *localhost* utilizando a Raspberry Pi.

O sistema foi projetado para executar um comando de controle por vez. Ele está protegido contra ataques a comunicação serial devido ao uso do algoritmo de MD5 e *salt*. A comunicação serial também é imune a interferência pois caso exista uma interferência durante o envio do comando, a placa de controle não executará o comando pois no cálculo do MD5 para confirmação da veracidade da mensagem, o comando não será executado devido a falha na mensagem. Apesar da segurança existir na comunicação, nenhum método foi implementado para proteger o controle a sensibilidade de ruídos.

O sistema de controle foi projetado para ser utilizado em qualquer situação que envolva controle digital, porém seria interessante utilizar uma placa com mais recursos de *hardware* do que o Arduino possa oferecer. O custo total do sistema foi de aproximadamente R$500,00.

5 Conclusão

A tecnologia de IoT aplicada à industria, ou IIoT, tem se mostrado uma alternativa viável para unificação das informações ao longo de toda a industria, principalmente quando se utiliza da infraestrutura de rede já disponível no local, reduzindo os custos de implementação.

Neste trabalho foi apresentado um protótipo de sistema de automação funcional composto por três estruturas: um servidor web, um protocolo de comunicação e uma placa de controle.

O servidor web funcionou como esperado e conseguiu enviar todas informações inseridas pelo o usuário para o sistema de controle. O servidor também efetuou a leitura das informações enviadas pelo controle sem nenhum contratempo.

A comunicação serial também cumpriu seu objetivo. Com os testes realizados no circuito RC foi possível observar o bom funcionamento do protocolo e do sistema de segurança implementado pelo MD5. Todos os comandos enviados foram interpretados corretamente pela placa de controle. Nos testes com simulação de falhas a placa de controle reconheceu o erro e não executou nenhuma operação insegura ou comando inválido.

O *firmware* da placa de controle também atingiu o seu objetivo: manter a funcionalidade de controle em tempo real ao mesmo tempo que interpretava corretamente os comandos enviados pelo servidor. O algoritmo escolhido, o MD5, foi implementado sem problemas de consumo excessivo de recursos, permitindo a segurança do sistema, inclusive com a utilização de um sistema de *salt*, simulando uma senha compartilhada.

A topologia apresentada para o sistema (web->comunicação->controle) se apresentou estável e de fácil a replicação para outros sistemas de automação ou controle industrial, principalmente por separar a infraestrutura de comunicação daquela responsável pelo controle do sistema. Isto reduz a probabilidade que erros e/ou invasões na camada de rede cheguem a contaminar o sistema de controle.

5.1 Trabalhos Futuros

Alguns aspectos do projeto não foram abordados em sua plenitude devido à falta de tempo hábil para desenvolvimento ou problemas técnicos obtidos durante sua aplicação e podem ser expandidos com maior profundidade em trabalhos futuros, entre eles:

- Desenvolvimento do website com mais recursos, inclusive monitoramento gráfico,

- O uso de outra placa de controle com mais recursos para acionamento de cargas,
- Utilização de um sistema de chave assimétrica, eliminando problemas de vazamento de senha e comandos não autorizados,
- Aprimoramento tanto da biblioteca PHP-Serial utilizada no servidor quanto a biblioteca MD5 utilizada no Arduino. Por serem desenvolvidas por terceiros, é possível terem alguns problemas que não foram notados durante o projeto.

Referências

AVOYAN, H. *Nginx vs Lighttpd*. 2012. <http://imasters.com.br/tecnologia/redes-e-servidores/nginx-vs-lighttpd/?trace=1519021197&source=single>. Acesso em 02 de Novembro de 2016. Citado na página 26.

BREI, T. M. *What is Industrial Automation*. 2013. <http://www.surecontrols.com/what-is-industrial-automation/>. Acesso em 05 de Novembro de 2016. Citado na página 13.

COSTA, C. da. *Projetando Controladores Digitais com FPGA*. [S.l.]: Novatec Editora, 2006. Citado na página 18.

CURVELLO, A. *Sistema Web com Raspberry Pi, Arduino, USB, Lighttpd e PHP*. 2015. <http://www.embarcados.com.br/sistema-web-com-raspberry-pi-e-arduino/>. Acesso em 04 de Setembro de 2016. Citado na página 29.

DAUGHERTY, P. et al. *Industrial Internet of Things*. 2016. <https://www.accenture.com/us-en/labs-insight-industrial-internet-of-things>. Acesso em 05 de Novembro de 2016. Citado na página 26.

DELAI, A. L. *Sistemas Embarcados: a computação invisível*. 2013. <http://www.hardware.com.br/artigos/sistemas-embarcados-computacao-invisivel/conceito.html>. Acesso em 04 de Setembro de 2016. Citado na página 21.

DESAI, N. *Industrial Internet of Things*. 2016. <http://itknowledgeexchange.techtarget.com/iot-agenda/the-worlds-of-it-and-ot-collide-within-the-industrial-internet-of-things/>. Acesso em 05 de Novembro de 2016. Citado na página 27.

FELIPE, E. M. C.; ALEXSANDER, P. S. E.; LEONARDO, A. M. *Sistema de gerenciamento multi-agentes utilizando visão computacional e sensoriamento embarcado*. Itajubá, 2014. Citado na página 20.

FELIPE, R. S. *Controle Digital Multivariável Utilizando RTOS Embarcado*. Itajubá, 2014. Citado 2 vezes nas páginas 16 e 21.

FOUNDATION, R. P. *RASPBERRY PI 2 MODEL B*. 2016. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Acesso em 04 de Setembro de 2016. Citado na página 21.

FREITAS, R. C. *Transmissão síncrona e assíncrona*. 2016. <https://www.passeidireto.com/arquivo/4366134/transmissao-sincrona-e-assincrona>. Acesso em 05 de Novembro de 2016. Citado na página 24.

GEORGITZIKIS, V. *ArduinoMD5*. 2015. <https://github.com/tzikis/ArduinoMD5>. Acesso em 07 de Novembro de 2016. Citado na página 32.

INSTRUMENTS, N. *Explicando a Teoria PID*. 2011. <http://www.ni.com/white-paper/3782/pt/>. Acesso em 04 de Setembro de 2016. Citado na página 16.

JIMBO. *Serial Communication*. 2012. <https://learn.sparkfun.com/tutorials/serial-communication>. Acesso em 05 de Novembro de 2016. Citado na página 24.

JUNIOR, W. A. A.; CRUZ, R. R. W. *INTRODUÇÃO À TRANSFORMADA Z*. Curitiba, 2010. Citado na página 19.

LUIZ, R. V. N.; VINICIUS, R. C. M.; VINICIOS, R. C. *Aplicação em tempo real controlada por Arduino*. Itajubá, 2015. Citado na página 22.

MADEIRA, D. *What is Industrial Automation*. 2016. <http://www.embarcados.com.br/iiot-industrial-internet-of-things/>. Acesso em 05 de Novembro de 2016. Citado na página 13.

NEVES, F. *Método de Tustin, transformando circuitos analógicos em equivalentes digitais para uso em sistemas embarcados*. 2015. <http://itknowledgeexchange.techtarget.com/iot-agenda/the-worlds-of-it-and-ot-collide-within-the-industrial-internet-of-things/>. Acesso em 06 de Novembro de 2016. Citado na página 18.

OGATA, K. *Engenharia de Controle Moderno*. 5ª edição. ed. [S.l.]: Pearson Education - Br, 2011. Citado na página 17.

OPPENHEIM, A. V.; SCHAFER, R. W. *Discrete-Time Signal Processing*. 3ª edição. ed. [S.l.]: Prentice Hall, 1989. Citado 2 vezes nas páginas 18 e 19.

RIBEIRO, M. A. Automação industrial. 1999. Citado na página 15.

RODARTE, C. *Criptografia MD5*. 2007. <http://www.devmedia.com.br/criptografia-md5/2944>. Acesso em 02 de Novembro de 2016. Citado na página 25.

ROUSE, M. *Internet of Things (IoT)*. 2016. <http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>. Acesso em 04 de Setembro de 2016. Citado na página 26.

SANCHEZ, R. *PHP-Serial*. 2014. <https://github.com/Xowap/PHP-Serial>. Acesso em 07 de Novembro de 2016. Citado na página 29.

SAS. *(IoT) Internet of Things*. 2016. <http://www.sas.com/pt_br/insights/big-data/internet-das-coisas.html>. Acesso em 04 de Setembro de 2016. Citado na página 26.

SILVA, J. A. F. da. *COMO FUNCIONA UM SERVIDOR WEB*. 2012. <http://www.portaleducacao.com.br/informatica/artigos/17165/como-funciona-um-servidor-web>. Acesso em 04 de Setembro de 2016. Citado na página 26.

SILVA, J. M. G. da. *Controladores Digitais*. 2000. <http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/node17.html>. Acesso em 04 de Setembro de 2016. Citado na página 18.

SILVA, J. M. G. da. *Controle em Malha Aberta*. 2000. <http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/node5.html>. Acesso em 05 de Novembro de 2016. Citado na página 15.

SILVA, J. M. G. da. *Controle em Malha Aberta*. 2000. <http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/node6.html>. Acesso em 05 de Novembro de 2016. Citado na página 15.

SILVEIRA, C. B. *O que é Automação Industrial*. 2013. <http://www.citisystems.com.br/o-que-e-automacao-industrial/>. Acesso em 04 de Setembro de 2016. Citado na página 15.

SOUZA, F. *Arduino - Saídas PWM*. 2014. <https://www.embarcados.com.br/arduino-saidas-pwm/>. Acesso em 06 de Novembro de 2016. Citado na página 23.

VERRY, T. *What is the Raspberry Pi?* 2012. <http://www.extremetech.com/computing/124317-what-is-raspberry-pi-2>. Acesso em 04 de Setembro de 2016. Citado na página 21.

WOOD, L. *Real-time computing: Gateway to the Internet of Things?* 2015. <http://www.computerworld.com/article/2973986/high-performance-computing/real-time-computing-gateway-to-the-internet-of-things.html?page=1>. Acesso em 05 de Novembro de 2016. Citado na página 21.

ZAMBADA, P. *'Internet das Coisas': entenda o conceito e o que muda com a tecnologia*. 2014. <http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entenda-o-conceito-e-o-que-muda-com-tecnologia.html>. Acesso em 04 de Setembro de 2016. Citado na página 26.

ANEXO A – Código da interface Web

```php
<?php
//rotinas para habilitar a exibicao de erros na pagina. Tire se nao quiser.
error_reporting(E_ALL & ~E_WARNING & ~E_NOTICE);
ini_set('display_errors', '1');

include "PhpSerial.php"; //import da biblioteca de serial com php
$read = "";
$salt = "tfgECO2016";

$serial = new phpSerial(); //Cria um novo objeto para comunicacao serial
//$serial->deviceSet("/dev/ttyS0"); //associa esse objeto com a serial do Arduino
//$serial->confBaudRate(57600); //configura baudrate em 9600
$serial->deviceSet("/dev/ttyACM0");
$serial->confBaudRate(9600);
$serial->confParity("none"); //sem paridade
$serial->confCharacterLength(8); //8 bits de mensagem
$serial->confStopBits(1); //1 bit de parada
$serial->confFlowControl("none"); //sem controle de fluxo
$serial->deviceOpen(); //abre o dispositivo serial para comunicacao

//Se receber 'a' via GET na Pagina
if(isset($_GET['a'])){
 $cmd = "digitalON";
 $str = "$" . "TFG" . "," . $cmd . "#";
 $strmd5 = $str . md5($str . $salt) . "\n";
 $serial->sendMessage($strmd5); //envia o caractere 'a' via Serial pro Arduino
 //echo printf "\n";
 sleep(1); //delay para o Arduino enviar a resposta.
 $read = $serial->readPort(); //faz a leitura da resposta na variavel $read
}

if(isset($_GET['c'])){
```

```
33  $cmd = "get";
34  $str = "$" . "TFG" . "," . $cmd . "#";
35  $strmd5 = $str . md5($str . $salt) . "\n";
36  $serial->sendMessage($strmd5); //envia o caractere 'a' via
       Serial pro Arduino
37  //echo printf "\n";
38  sleep(1); //delay para o Arduino enviar a resposta.
39  $read = $serial->readPort(); //faz a leitura da resposta na
       variavel $read
40 }
41
42 //Se receber 'd' via GET na pagina
43 if(isset($_GET['d'])){
44  $cmd = "digitalOFF";
45  $str = "$" . "TFG" . "," . $cmd . "#";
46  $strmd5 = $str . md5($str . $salt) . "\n";
47  $serial->sendMessage($strmd5); //envia o caractere 'a' via
       Serial pro Arduino
48  sleep(1); //delay para o Arduino enviar a resposta
49  $read = $serial->readPort(); //faz a leitura da resposta na
       variavel $read
50 }
51
52 if(isset($_GET['b'])){
53  $cmd = "digitalON";
54  $str = "$" . "TFG" . "," . $cmd . "#";
55  $strmd5 = $str . md5($str . $salt - 1) . "\n";
56  $serial->sendMessage($strmd5); //envia o caractere 'a' via
       Serial pro Arduino
57  sleep(1); //delay para o Arduino enviar a resposta
58  $read = $serial->readPort(); //faz a leitura da resposta na
       variavel $read
59 }
60
61
62 if(isset($_GET['ref'])){
63  $ref = $_GET['ref1'];
64  $kp = $_GET['Kp'];
65  $ki = $_GET['Ki'];
66  $kd = $_GET['Kd'];
67  $cmd = $ref . "&" . $kp . "+" . $ki . "*" . $kd;
68  $str = "$" . "TFG" . "," . $cmd . "#";
```

```
69  $strmd5 = $str . md5($str . $salt) . "\n";
70  $serial->sendMessage($strmd5); //envia o caractere 'a' via
        Serial pro Arduino
71  sleep(1); //delay para o Arduino enviar a resposta
72  $read = $serial->readPort(); //faz a leitura da resposta na
        variavel $read
73  }
74
75  $serial->deviceClose(); //encerra a conexao serial
76
77  ?>
78
79  <html>
80  <head>
81  <link rel="stylesheet" type="text/css" href="tfg.css">
82  <meta http-equiv="Content-Type" content="text/html;charset=utf-8"
        />
83  <meta charset="UTF-8">
84  </head>
85
86
87  <body style="margin: 0 0 0 0;">
88  <div class="art1">
89      <div class="dropdown" style="float:left;">
90        <button class="dropbtn">About</button>
91        <div class="dropdown-content" style="left:0;">
92          <a>Trabalho Final de Graduacao<br>Andre Luiz de Souza
              Antonieto<br>Engenharia da Computacao<br>Orientador:
              Rodrigo Almeida<br>2016</a>
93        </div>
94      </div>
95      <div class="cent">
96          <h1 style="font-family: Verdana;font-size: 50px;"><center
              > Automacao Industrial <img hspace="20" src="
              logo_unifei.png" width="150" height="150" align="center
              "></center></h1>
97      </div>
98  </div>
99  <div class="art2">
100     <center>
101             <br><br><br>
102
```

```
103             <input type="button"
104              onclick="location.href='/tfg.php?a=1'"
105              value="Digital ON"
106              />
107
108             <input type="button"
109              onclick="location.href='/tfg.php?d=1'"
110              value="Digital OFF" />
111
112              <input type="button"
113              onclick="location.href='/tfg.php?b=1'"
114              value="MD5 False" />
115
116              <input type="button"
117              onclick="location.href='/tfg.php?c=1'"
118              value="GET"/>
119
120             <br><br><br>
121     </center>
122
123         <form name="form2" action="" method="">
124             <center>
125          <font size="5" style="font-family: Verdana; font-weight:
               bold;">Kp : </font><input type="text" value="" name="
               Kp" style="width: 100px;height: 25px" required>
126          <font size="5" style="font-family: Verdana; font-weight:
               bold;">Ki : </font><input type="text" value="" name="
               Ki" style="width: 100px;height: 25px" required>
127          <font size="5" style="font-family: Verdana; font-weight:
               bold;">Kd : </font><input type="text" value="" name="
               Kd" style="width: 100px;height: 25px" required>
128          <br><br>
129          <font size="5" style="font-family: Verdana; font-weight:
               bold;">Reference : </font><input type="text" value=""
               name="ref1" style="width: 100px;height: 25px" required
               >
130          <br><br>
131          <input type="Submit" name="ref" value="Send">
132          </center>
133         </form>
134
135
```

```
136     <br><br>
137     <center>
138             <font size="5" style="font-family: Courier; color: #
                    F73333; font-weight:bold;"><?php echo $read?></font
                    >
139     </center>
140 </div>
141 </body>
142 
143 </html>
```

ANEXO B – Código Fonte da Placa Controladora

```
1  // include the library code:
2  #include <TimerOne.h>
3  #include <MD5.h>
4  #include <string.h>
5  #include <ctype.h>
6  #include <stdio.h>
7
8  boolean flag;
9  int k=0;
10 int n=0;
11 int nref=255;
12 int nkp=1;
13 int nki=10;
14 int nkd=0;
15
16
17 //Controlador PID e Tempo Real
18 int ledPin = 11;
19 int analogPin = 3;
20 int y0,y1,y2,e0,e1,e2;
21 double kp=1, ki=10, kd=0   , T=0.001, ref=255;
22
23 void PID(void){
24   int an = analogRead(0);
25   digitalWrite(8,HIGH);
26   e2=e1;
27   e1=e0;
28
29   e0=ref-(an/4);
30   y0=y1;
31   y1=y2;
32   y2 = y0 + (kp * (e0 -e2)) + (ki * (e0 + (2*e1) + e2) * T/2) + (
        kd * (e0 - (2*e1) + e2) * 2/T);
33   if(y0 > 255){
34
35     analogWrite(ledPin,255);
```

```
36    }else if(y0 < 0){
37      analogWrite(ledPin,0);
38    }else{
39      analogWrite(ledPin,y0);
40    }
41    digitalWrite(8,LOW);
42  }
43
44  void setup() {
45    Serial.begin(9600); //starts serial communication
46    pinMode(0,INPUT);
47    pinMode(ledPin, OUTPUT);
48    pinMode(8, OUTPUT);
49    pinMode(7, OUTPUT);
50    Timer1.initialize(1000);
51    Timer1.attachInterrupt(PID);
52  }
53
54  #define pkgSIZE 70
55  //+1 -> \0
56  #define md5SIZE (32+1)
57  #define saltSIZE (10+1)
58  static char salt[] = "tfgECO2016";
59
60  #define msgSIZE (pkgSIZE - md5SIZE + saltSIZE)
61
62
63  #define HEADER 4
64  #define cmdSIZE (msgSIZE - HEADER)
65
66  int pos;
67  char pkg[pkgSIZE];
68  char pkgmd5[md5SIZE];
69  char pkgmsg[msgSIZE];
70  char pkgcmd[cmdSIZE];
71
72  boolean StringSerial(){
73    char ch;
74    while(Serial.available() > 0){
75      ch = Serial.read();
76      Serial.print(ch);
77      if(ch == '$'){
```

```
78        pos=0;
79      }
80      pkg[pos] = ch;
81      pos++;
82      if(ch == '\n'){
83        int i;
84        int md5POS;
85        pkg[pos]='\0';
86
87        //find msg
88        for(i=0;i<pkgSIZE;i++){
89          pkgmsg[i] = pkg[i];
90          if(pkg[i] == '#'){
91            md5POS = i+1;
92            break;
93          }
94        }
95        for(i= 0;i<saltSIZE;i++){
96          pkgmsg[i+md5POS] = salt[i];
97        }
98
99        //find md5
100       for(i = md5POS;i<pkgSIZE;i++){
101         if(pkg[i] == '\n'){
102           break;
103         }
104         pkgmd5[i-md5POS] = pkg[i];
105       }
106       pkgmd5[i-md5POS] = '\0';
107
108
109       //find cmd
110       for(i=5;pkg[i] != '#';i++){
111         pkgcmd[i-5] = pkg[i];
112       }
113       pkgcmd[i-5] = '\0';
114
115       pos=0;
116       return true;
117     }
118     if(pos > pkgSIZE){
119
```

```
120       Serial.print("Send the message again!");
121       pos=0;
122       return false;
123     }
124   }
125
126   return false;
127 }
128
129 void loop() {
130
131   if(StringSerial()){
132
133     unsigned char *hash=MD5::make_hash(pkgmsg);
134     char *md5 = MD5::make_digest(hash,16);
135
136     flag = true;
137     for(int i=0; i<32; i++){
138       if(pkgmd5[i] != md5[i]){
139         flag = false;
140       }
141     }
142     if(flag){
143
144       if(!strcmp(pkgcmd,"digitalON")){
145         digitalWrite(7,HIGH);
146       }
147       else{
148         if(!strcmp(pkgcmd,"digitalOFF")){
149           digitalWrite(7,LOW);
150         }
151         else{
152           if(!strcmp(pkgcmd,"get")){
153             Serial.print(" Kp = ");Serial.print(kp);
154             Serial.print(" Ki = ");Serial.print(ki);
155             Serial.print(" Kd = ");Serial.print(kd);
156           }
157           else{
158             k=0;
159             n=0;
160             while(pkgcmd[k] >= '0' && pkgcmd[k] <= '9'){
161               n=n*10 + (pkgcmd[k] - '0');
```

```
162                 k++;
163                 nkd=n;
164                 if(pkgcmd[k] == '&'){
165                   k++;
166                   nref=n;
167                   n=0;
168                 }else{
169                   if(pkgcmd[k] == '+'){
170                     k++;
171                     nkp=n;
172                     n=0;
173                   }else{
174                     if(pkgcmd[k] == '*'){
175                       k++;
176                       nki=n;
177                       n=0;
178                     }
179                   }
180                 }
181               }
182               ref=nref;
183               kp=nkp;
184               ki=nki;
185               kd=nkd;
186             }
187           }
188         }
189       }
190       else{
191         Serial.print("MD5 MISMATCH");
192         Serial.print(" Send the message again!");
193       }
194
195     }
196   }
```